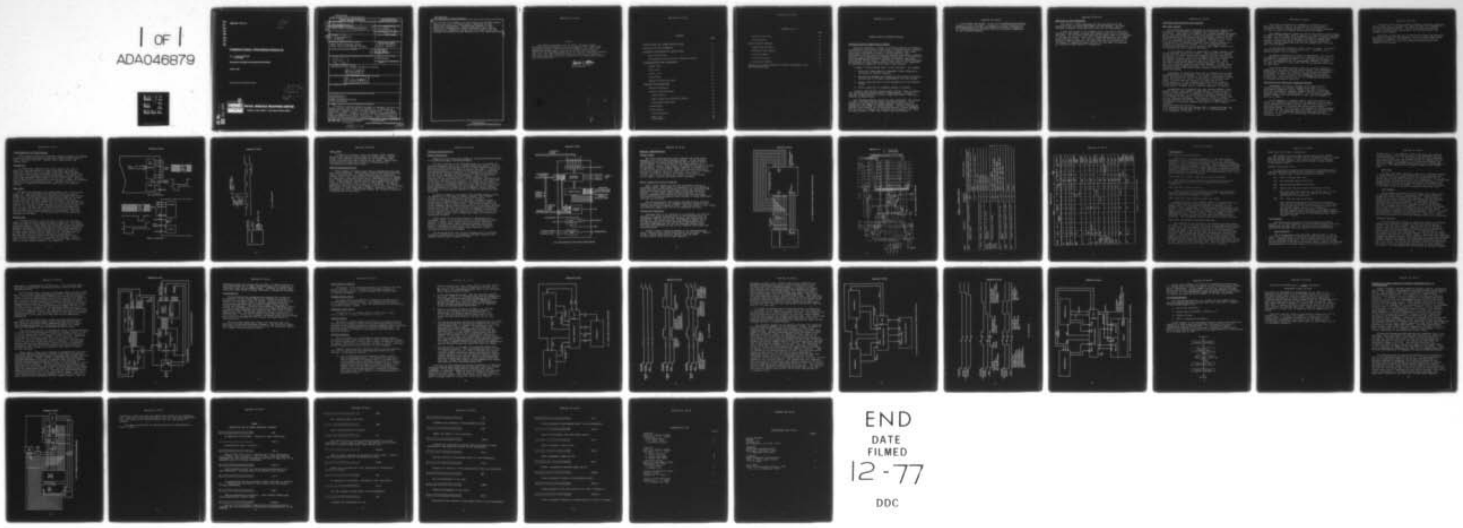
AD-A046 879 NAVAL SURFACE WEAPONS CENTER WHITE OAK LAB SILVER SP--ETC F/G 9/5
COMMON SIGNAL PROCESSING MODULES.(U)
MAY 76 C N PRYOR, L S HAYNES
UNCLASSIFIED NSWC/WOL/TR-76-175 NL

1 OF 1
ADA046879



END
DATE
FILMED
12-77
DDC

NSWC/WOL TR 76-175

12
B.S.

COMMON SIGNAL PROCESSING MODULES

BY C. NICHOLAS PRYOR
L. S. HAYNES

ORDNANCE SYSTEMS DEVELOPMENT DEPARTMENT

10 MAY 1977

Approved for public release; distribution unlimited.

NAVAL SURFACE WEAPONS CENTER

Dahlgren, Virginia 22448 • Silver Spring, Maryland 20910

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) NSWC/WOL/TR-76-175	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Common Signal Processing Modules,		5. TYPE OF REPORT & PERIOD COVERED (9) Final Report,
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) C. Nicholas Pryor L. S. Haynes		8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Surface Weapons Center
White Oak, Silver Spring, MD 20910
		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
		0; MAT-03L-000/ZF61-001;
		ZF61-001; S2519-002;

11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE (11) 10 May 1976
		13. NUMBER OF PAGES 43
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 48p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

(16) F61001

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(17) ZF61001

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

microcontroller,
microprocessor,
signal processing modules
microprogramming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Currently, special purpose hardware is designed for each new weapon system, with little or no capability of being easily modified or adapted to other similar applications. This paper describes a group of small, low power signal processing modules, with a compatible, flexible structure for intercommunication of data and command and control information. Instead of designing systems "from scratch" for a unique application, system designers →

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391 596

1B

will be able to configure and then program the standard digital modules into small or large systems, optimized to their particular requirements. Implementing hardware in this manner will substantially reduce development time, testing requirements, logistics problems, and life cycle costs, and increase the versatility and reliability of the resulting product.

PREFACE

This report describes a set of small, low power signal processing modules capable of being configured into a wide range of weapons systems. This work was performed by the Digital and Signal Processing Branch of the Naval Surface Weapons Center. The effort was supported primarily as an internal exploratory development task, work unit MAT-03L-000/ZF61-001, and certain aspects of the work herein described were supported by the mobile mine project, work unit S2519-002.

Edward C. Whitman
EDWARD C. WHITMAN
By direction

CONTENTS

CONTENTS (Cont.)

COMMON SIGNAL PROCESSING MODULES

JUSTIFICATION FOR COMMON MODULE DESIGN

There are currently a number of programs proposed or presently in exploratory development in the areas of advanced mines, torpedoes, surveillance sonobuoys, and remote sensors systems whose signal processing requirements overlap to varying degrees. The requirements include case orientation sensing, correlation, spectrum analysis, and post-analysis processing for automatic line detection, line integration and tracking, bearing computation, and target tracking and classification. Historically, special purpose hardware was designed for each weapons system, with little or no capability of being easily adapted to other similar applications.

A number of shortcomings exist in this approach. For example:

1. There is a large amount of redundant effort expended in the design of similar hardware.

2. Methods and hardware for simulation and testing of weapons systems must be designed independently for each system.

3. Failure analysis must be done separately for each weapons system.

4. Growth capability of hardware systems is minimal.

In short, many devices which require similar types of complex data handling are designed entirely independently. Increased cost due to duplication of effort, longer development times, and inferior products result.

The standard modules, which this paper describes, are a group of small, low power digital signal processing modules. These modules will be capable of being combined in various ways to implement a wide range of weapons systems. Instead of designing systems "from Scratch" for a unique application, system designers will be able to configure the standard digital modules into small or large systems, to meet their particular requirements.

The primary requirement of any such standardized processing elements is that they have a compatible, flexible structure for intercommunication of data and command and control information. The intercommunication structure will be continually emphasized in the following chapters.

SELECTION OF CMOS TECHNOLOGY

The choice of CMOS technology for the microcontroller was dictated by the low power requirement for the modules. No other currently available technology could have met the requirement.

A read only memory is an integral part of the microcontroller. Presently a MOS memory is being used because there are no convenient CMOS ROMS. We can expect more practical low power ROMS to be available in the next few years. Until then, in many applications, the higher power ROMS may be used. If they are turned off when the ROM is not actually doing calculations, the power they use is proportional to the amount of computation done, and will be very small for problems not requiring large amounts of computation.

SUBSYSTEM INTERCONNECTION PHILOSOPHIES

Wait Line Concept

The processing modules discussed in the previous chapter must be capable of being easily configured into relatively complex structures. Many tasks, for example, will require several processing units, each performing part of the total amount of work to be done. The synchronization and communication philosophy followed in the design of the overall system is extremely important if the system is to be expandable in size and function. If the initial formulation is not adequately general, then complex systems will be messy or impossible to design. If the configuration and communication structures are too general, then simple systems will be unnecessarily slow, complex, and costly.

In the simplest systems or subsystems of larger systems, there is one processor which is the "boss". This processor is entirely responsible for controlling every operation done by every device on every clock cycle. The control signals from this processor may, for example, connect directly to the gates, and busses within the module, controlling the data flow in a direct way. This processor always knows whether a particular module is busy or not, since no other processor can issue commands. The QED (Quick and Easy Design) program (reference (1)) at the Navy Electronics Laboratory Center, San Diego, deals primarily with this level of system structures.

Subsystems, as described above, may be combined into systems employing levels of processors. At any level, there is only one "boss" controlling the devices at its level. Now however, a given device may itself be a "boss" of the devices at the next lower level. The boss does not now know the exact status of a device because it in turn depends on the devices it controls. A synchronization scheme is now needed to see if a "boss" can accept another command, and if a desired operation is complete.

Subsystems can be combined into more complex systems, where several "bosses" can command a single module. Resources are now shared by many processes. A memory, for example, may be used by several processes. A synchronization scheme is needed to see if the memory is available. Priorities are needed to insure that the most important processes receive service first, yet every process receives some service. In addition, a communication network is required to multiplex commands and data to or from a given resource.

[1] 2175 Program Quick and Easy Design "QED" of Systems Through High Level Functional Modularity, TR 1904, N. L. Tinkelpaugh and D. C. Eddington, 28 Jan 1974.

The actual interconnection hardware is dependent on the particular system. What is of concern here is the command structure which will enable systems to be conveniently configured at both the lowest and highest levels within the system.

The command structure chosen for use in the modules being designed is based on a WAIT LINE. When a command is issued to several processes, the Wait line is immediately grounded, forcing the requesting process to wait. When all of the processes receiving the command have completed enough processing to permit the requesting process to proceed, (having removed any data from the busses, etc.) then the Wait line is raised.

In the simplest subsystem, no Wait line is needed. The "boss" processor knows what is busy and what isn't. The Wait line is merely shorted to the Hi supply.

In systems consisting of levels of subprocesses, if a process is commanded to perform an operation, it grounds its Wait line stopping the commanding processor. It will unground the Wait line when it is permissable for the commanding processor to proceed. If there are several levels, there will be one separate Wait line for each level.

In the most complex systems, there are several processors competing for services from the same processes. The communication network must contain the necessary bus structures and priority arbitrators to insure that only one command actually reaches a process at a time, and that if the commanded process grounds its Wait line , then the Wait line of the commanding process will be grounded halting its operation.

Horizontally and Vertically Organized Systems

Subsystems, and finally systems can be organized with either horizontal or vertical command structures. In a horizontally organized system, command lines are not decoded. Each line represents a separate command, and as a result, many commands may be given simultaneously. In a vertically organized system, command lines are decoded, so that only one command is given at a time.

In the design of a complete system, both horizontally and vertically organized controllers will be employed. At the lowest level, primarily horizontal control word bits or fields will be connected directly to the gates and control points within a processing module. At this level, the sequence of control signals will cause the data flow necessary to perform the processing task required. As the commanding controller knows exactly what the status is of what it is controlling, the Wait line will not normally be used.

At higher levels within systems, primarily vertically organized control lines will be decoded into specific high level commands to the lower processing modules. The Wait line will be used here to stop the higher level module until the receiving module can respond to the command.

Naturally, controllers can be partially vertical and partially horizontal, having several decoders, and hence permitting several simultaneous commands. The Wait line will be used to stop the commanding process until the receiving processing modules can respond to the request.

INTERCONNECTION PORT DEFINITIONS

Three kinds of ports are considered among the digital processing modules, and a given module may have one or more of each type of port. The three types are: Output Ports, Input Ports, and Control Ports.

Output Port

An output port consists of an Output Enable (OE) line, a Wait line, and some number of Output Data lines, as shown in Figure 1a. The OE line is an input to the port, while the Wait line and the Data lines are outputs of the port. The output data lines are to be implemented with tri-state drivers, which are in their high impedance state whenever the OE line is high. When the OE line goes low, the output data lines are put in their low impedance state to output the appropriate data. The Wait line output is connected to commanding processes as described in the previous section.

Input Port

An input port consists of an Input Enable line, a Clock line, a Wait line and some number of Input Data lines, as shown in Figure 1b. Each input data line connects to an input latch in the module which holds some previous value while either the IE line or the Clock line is high. When both the clock line and the IE line are low, the latch passes the input data. The new data is held by the latch when either the clock or the IE line returns high. Normally the IE line is stable during any period that the Clock line is low. The Wait line is normally left in the high state and is pulled down whenever the IE and Clock line have gone low and the module has not yet accepted a new data input. Pulling this line low delays the controlling device until the module has accepted the new data.

Control Port

A Control Port has a Clock Out line, a Wait line, and some number of Control Level outputs as shown in Figure 2. The control levels are used as Output Enable and Input Enable lines for various controlled Output or Input ports and for other control information (generally going to various data inputs of controlled modules) as desired. The control outputs change only during times the clock is high and are stable when the clock is low. The Wait line is an input which stops the clock whenever it is pulled low. When the Wait line is pulled low by some external device, the clock proceeds normally until it reaches its LO state. Then it stops in the LO state until the Wait line returns high, at which time it resumes normal operation. See Figure 2.

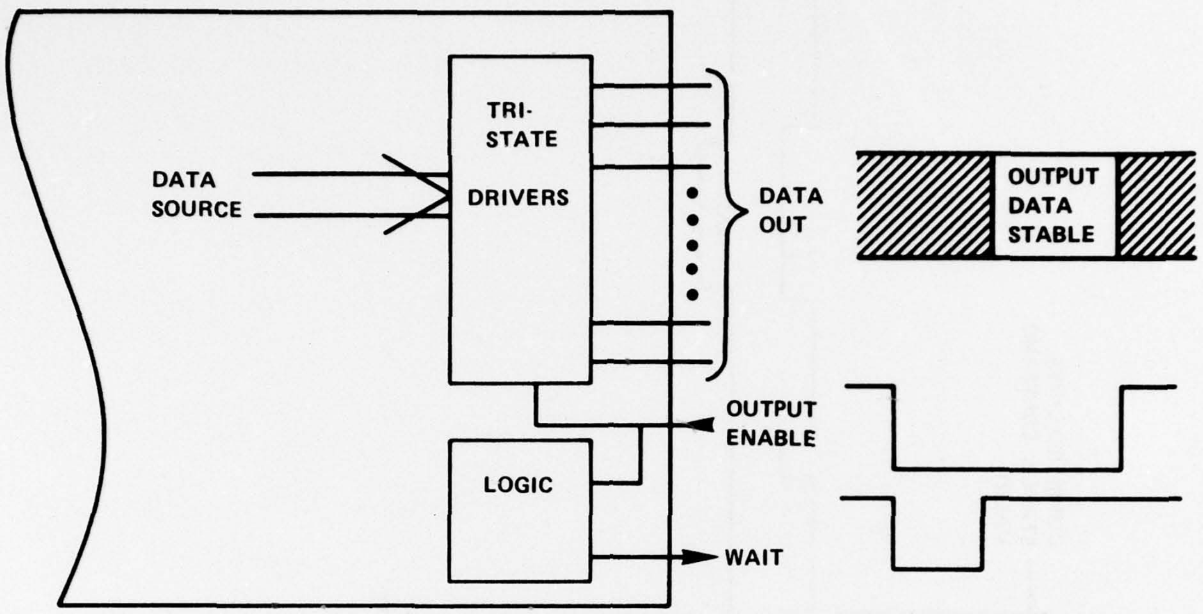


FIG. 1a OUTPUT PORT

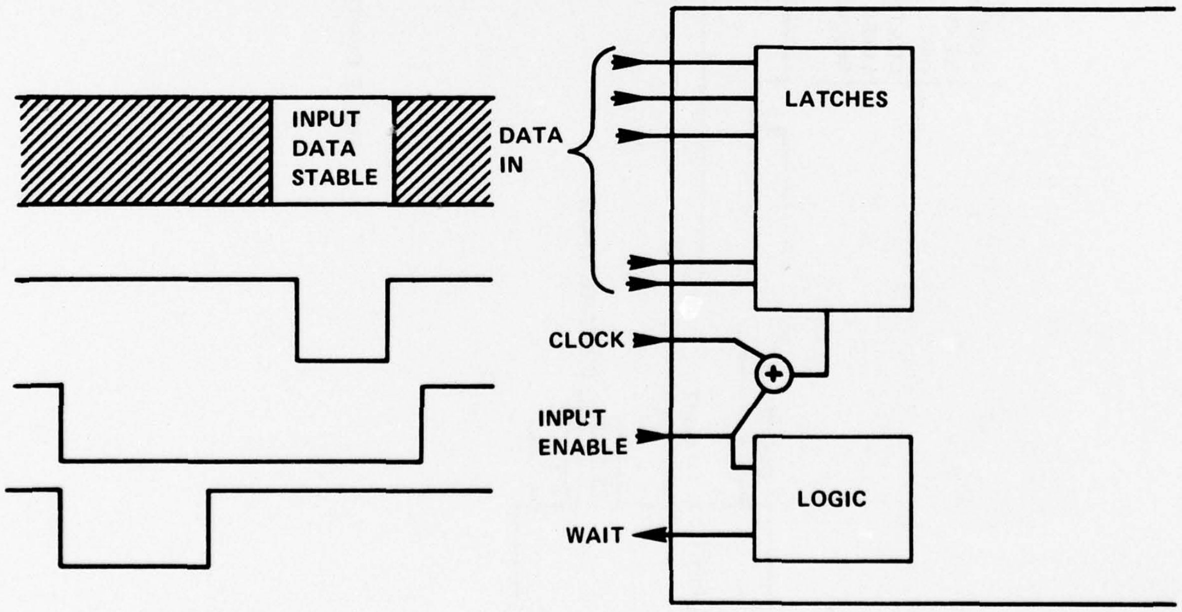


FIGURE 1b. INPUT PORT

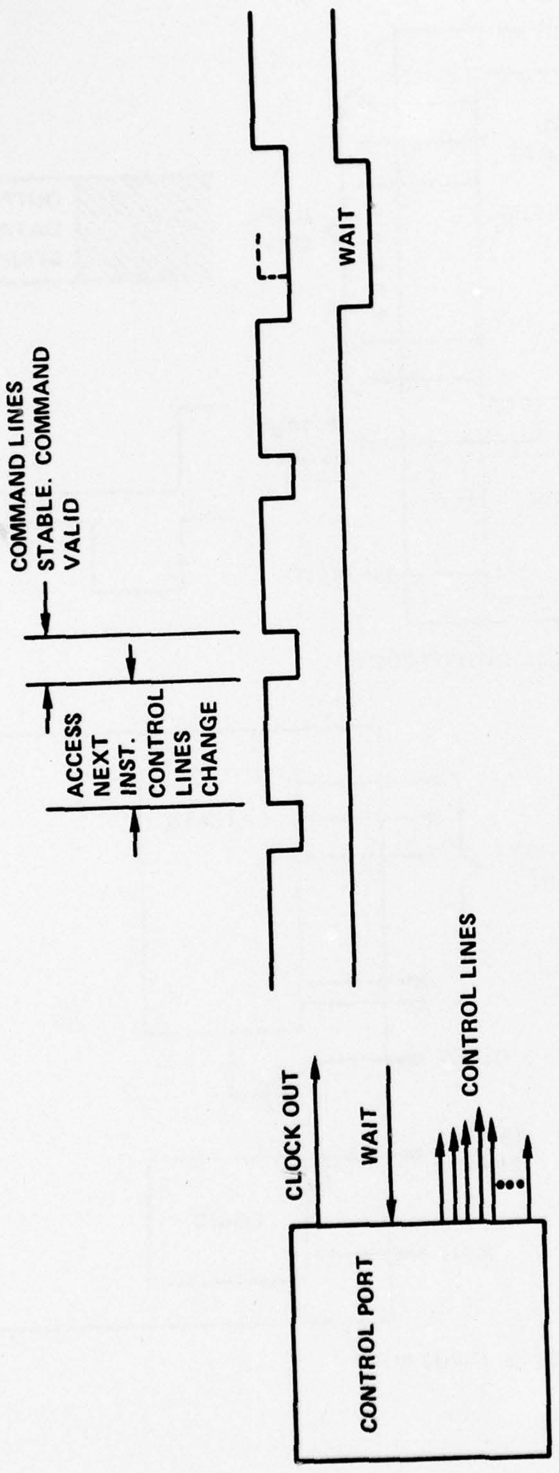

CONTROL PORT
CLOCK OUT
WAIT
CONTROL LINES
ACCESS NEXT INST. CONTROL LINES CHANGE
COMMAND LINES STABLE. COMMAND VALID
WAIT

FIG. 2 CONTROL PORT

Other Lines

In addition to Output, Input, and Control ports, modules may also have discrete status input or output lines. Status output lines indicate some discrete bit of information from within the module and may be active at all times. Status input lines provide a module with a means of testing discrete information from some other module.

Reduced Flexibility Ports

Input, Output, or Control ports may be implemented with less than full flexibility. For example a control port which never has need to delay an input or output operation may be implemented without a Wait line. Output ports for functions which will always be connected to dedicated inputs may be implemented with no Enable line. Input ports not needing latches may be implemented without Clock or Input Enable line, or only an Input Enable line may be required if strobing of input data is not required. For modules having more than one port, some signals such as data input or output, Wait lines, or Clock lines may be common. However it should be recognized that any of these steps reduces the ultimate flexibility of application of the device.

STANDARD MICROCONTROLLER

General Definition

Figure 3 is the conceptual diagram of the microcontroller module of the Standard Signal Processing modules.

The basic function of the microcontroller is to provide the operation sequence control for a larger processing system. This is done by controlling the address fed to a control memory, from which microinstructions for the entire processor are read. A portion of each microinstruction is used by the microcontroller to determine the address of the next operation, the remainder is available for control of the remainder of the processor. The microcontroller must be capable of providing basic timing signals to the remainder of the processor and of testing processor information for conditional branches in the sequence. In addition, some means must be provided for recognizing requests from higher level elements in the system or interrupts from within the system being controlled. The microcontroller defined here is an eight bit slice capable of being implemented in a single 48 pin integrated circuit package. Any number of slices may be combined to yield an arbitrarily wide controller.

As shown in Figure 3, the basic 8 bit Program Counter (slice) can address up to 256 words of Control Memory. This Program Counter can either be incremented during an operation to permit microinstructions to be executed in normal sequence, loaded from an address input field to permit arbitrary jumps, or it may interact with a Control Stack. The Stack is used to hold return addresses during subroutine execution or interrupt handling and may also be used as a loop counter for repeated executions of blocks of microcode. The function to be performed is determined by four Function Code inputs (extended in some cases by five bits of the Address Inputs), and a single Test input is provided for conditional branches.

Basic timing of the microcontroller is generated by an Oscillator input. The Oscillator drives a Clock Generator which produces an output clock signal used to drive any controlled devices. A Clock Enable input determines whether the Clock Output is generated on each cycle, and a Wait line returning from the controlled devices permits these devices to delay instruction execution as necessary.

The Microcontroller chip is also provided with an Interrupt capability, permitting it to recognize interrupt requests from external devices and to respond to them.

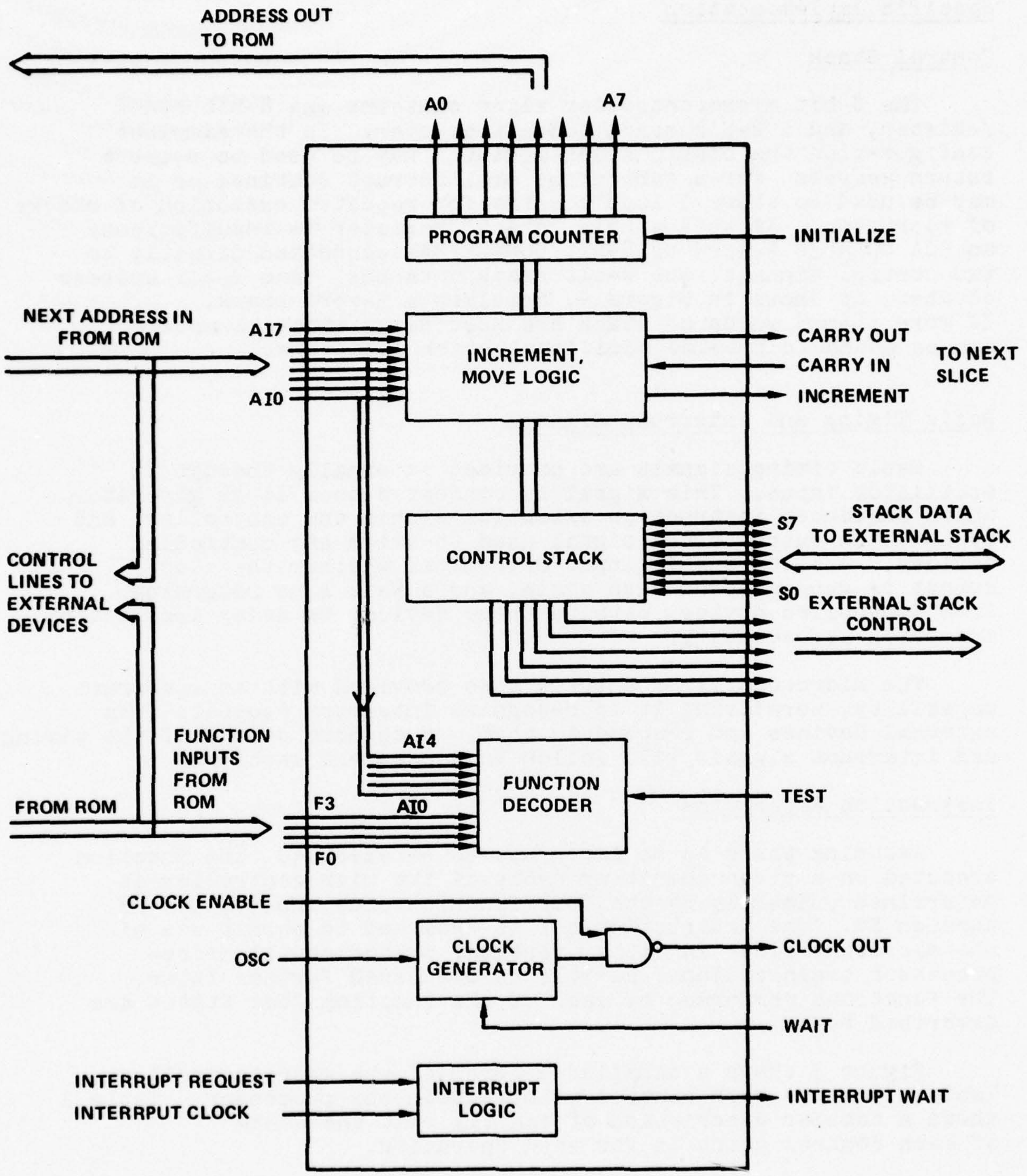


FIG. 3 MICROCONTROLLER FUNCTIONAL BLOCK DIAGRAM

Specific Implementation

Control Stack

The 8-bit microcontroller slice contains one 8-bit stack register, and a 2-bit stack address counter. In the simplest configuration the single stack register may be used to store a return address for a subroutine or interrupt routine; or it may be used to store a loop counter for repeated execution of blocks of microcode. If this single "stack" register is insufficient, an RCA CD 4036 4-word by 8-bit COSMOS RAM connected directly to two control signals, the 8-bit stack data bus, and 2-bit address counter, as shown in Figure 4, provides a 5-word stack. If more than 5 words of stack are needed, an external module can be added to provide additional stack locations.

Basic Timing and Interrupt Signals

Basic timing signals are provided externally through an oscillator input. This signal is connected to a clock generator which sequences instruction execution within the controller, and produces an output clock signal used to drive any controlled devices. A clock enable input determines whether the clock output is generated on each cycle, and a Wait line returning from controlled devices permits these devices to delay instruction execution as necessary.

The microcontroller chip is also provided with an interrupt capability, permitting it to recognize interrupt requests from external devices and respond to them. Much more detail of the timing and interrupt signals will follow in subsequent sections.

Instruction Repertoire

Assuming there is no interrupt to be executed, the function executed on a given operating cycle of the microcontroller is determined primarily by the four function code input lines F3 through F0. The instruction set is designed to permit use of the microcontroller in either vertical or horizontal microprocessor organizations, as will be discussed further later. The functions performed by each of the Function Code states are described below.

Figure 5 shows a detailed diagram of the microcontroller. Table 1 defines each control point within the processor. Table 2 shows a tabular description of exactly what the state of each control point is for each operation.

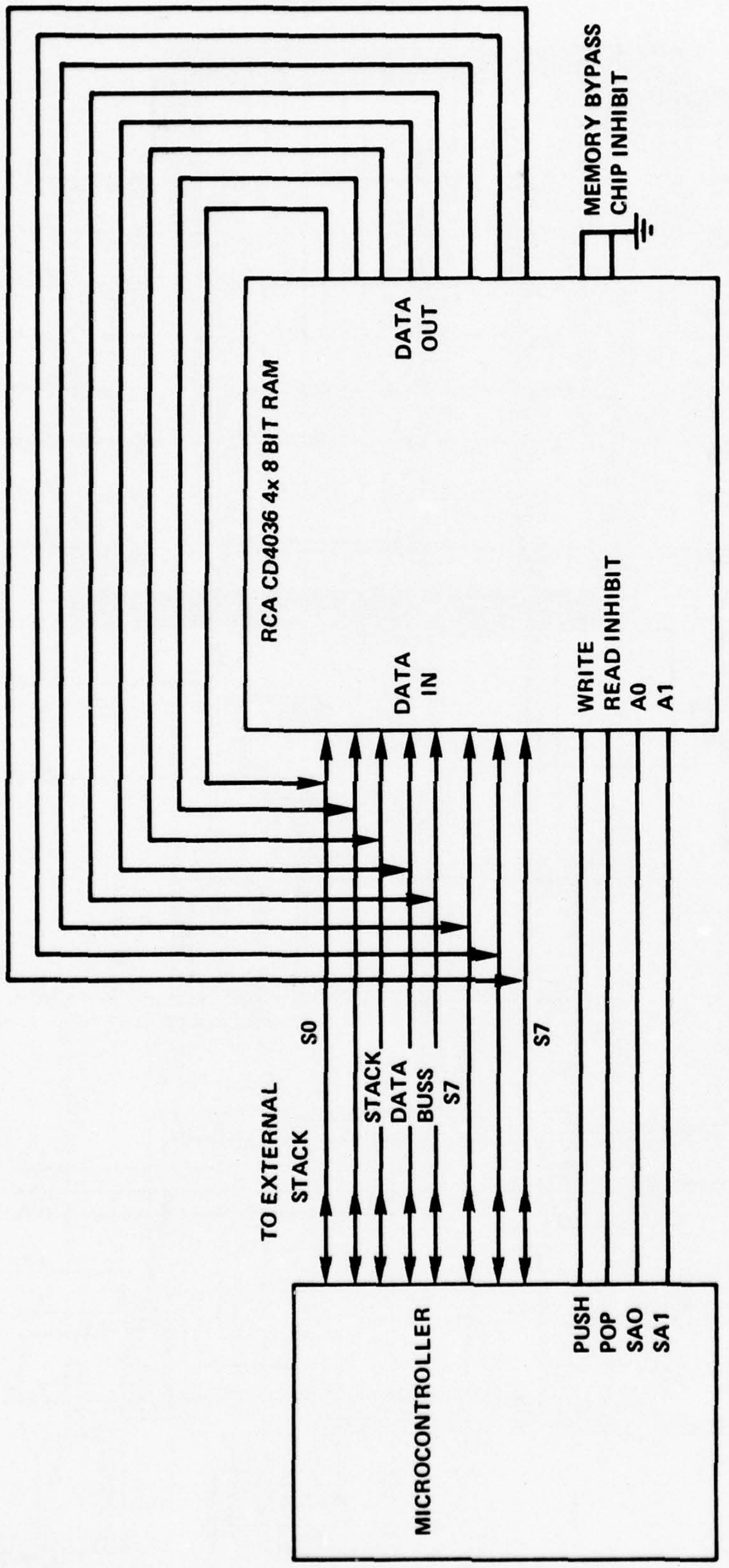


FIG. 4 CONNECTION TO EXTERNAL MEMORY FOR 5 WORD STACK

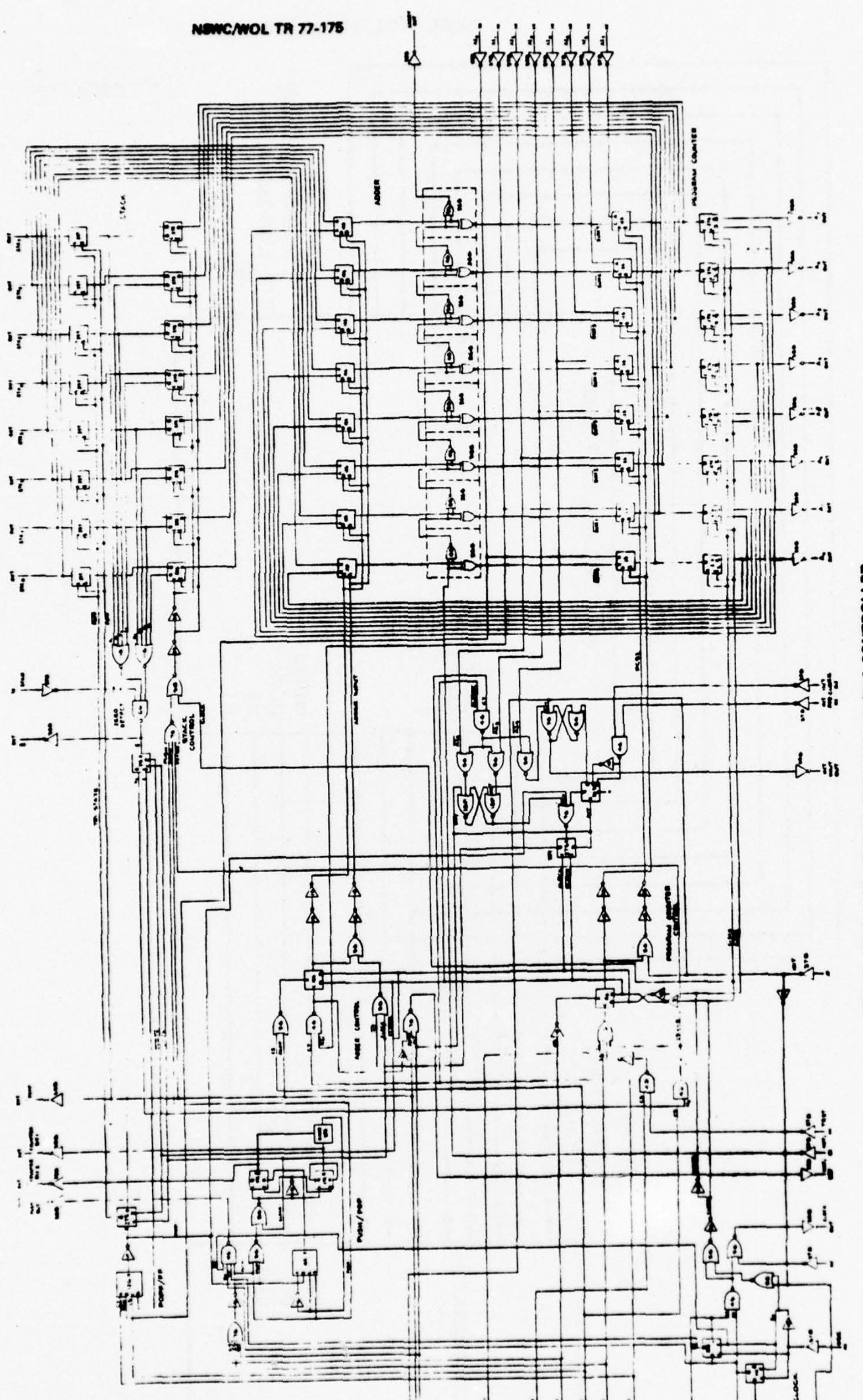


FIG. 5 DETAILED CIRCUIT DIAGRAM OF THE CONTROLLER

TABLE 1 CONTROL POINTS

NAME	FUNCTION	POSSIBLE VALUES	FUNCTION/VALUE
PCSI	Controls input to program counter and to stack	00	No input to program counter or stack
		01	Input to PC and stack from output of adder
		10	Input to PC and stack from address inputs AIO - AI7
3 State	Enable signal for stack data buss	0	Output enabled
		1	Output disabled
Stack Clock Enable	Enables clock to top stack register	0	Clock disabled
		1	Clock enabled
Adder Input Select	Controls input to adder	00	No input to adder
		01	Program counter gated to adder
		10	Stack output gated to adder
HOT 1	Controls first bit carry in for adder	1	Carry bit off - do not add one
		0	Carry bit on - add one
PUSH	Signals external device to "PUSH"	0	-
		1	"PUSH"
POP	Signals external device to "POP"	0	-
		1	"POP"

CONTROL POINT VALUES

	CLOCK								$\overline{\text{CLOCK}}$				
OPERATION	PCSI BD	3 STATE	STACK CLOCK ENABLE	ADDER INPUT BD	HOT 1	PUSH OUT	POP	INC-DEC STCNT	PCSI BD	ADDER INPUT DB	HOT 1	SCT POPF	OPERATION
1xxx	dd	0	0	dd	d	0	0	0	ADDER 01	PC 01	1	0	PC←PC+1
JUMP 0001	dd	0	0	dd	d	0	0	0	AI 10	dd	d	0	PC←AI
JTL TH	dd	0	0	dd	d	0	0	0	ADDER 01	PC 01	1	0	PC←PC+1
0011 IL	dd	0	0	dd	d	0	0	0	AI 10	dd	d	0	PC←AI
PUSH 0100	AI 10	0	1	dd	d	CLOCK PUSH $A_xB_xC_x$	0	0	ADDER 01	PC 01	1	0	STCNT←STCNT+1 STKT←AI PUSH PC←PC+1
TNIJ Z	dd	0	0	dd	d	0	1	1	ADDER 01	PC 01	1	1	SET POPF; STCNT←STCNT -1 PC←PC+1
0101 NZ	ADDER 01	0	1	STKO 01	1	0	0	0	AI 10	dd	d	0	STKT←STKT+1 PC←AI
POPF	ADDER 01	SWITCH OPEN 1	1	STACK 01	0	0	0	0	ADDER 01	PC 01	0	0	POP STACK PC←PC
JSUB 0110	ADDER 01	0	1	PC 10	1	1 (as above)	0	1	AI 10	dd	d	0	STACK←PC+1 STCNT←STCNT+1 PC←AI
C7←AI0=POP AI0 AI1..AI4 0111 01000	ADDER dd	0	0	dd	1 (if return=0)	0	1	1	ADDER 01	PC 01	1 (if return=0)	1	SET POPF; PC←PC+1 STCNT←STCNT-1
C7←AI1=return AI0 AI1..AI4 0111 10000	dd	0	0	dd	d	0	0	0	ADDER 01	STKO 10	regardless of C7 options	0	PC←STACK T
C7←AI2 reset int enb	dd	0	0	dd	d	0	0	0	ADDER 01	PC 01	1	0	PC←PC+1
C7←AI3	dd	0	0	dd	d	0	0	0	ADDER 01	PC 01	1	0	PC←PC+1
C7←AI4	dd	0	0	dd	d	0	0	0	ADDER 01	PC 01	1	0	PC←PC+1
INT	ADDER 01	0	1	PC 10	0	1 (as above)	0	1	ADDER 01	OUTPUT ALL 1'S [01010101] 00	0	0	RESET INT. ENABLE STACK←PC PC← 01010101 STCNT←STCNT+1
INIT	PC ← ALL 1'S	0	0	PC 10	1	0	0	0	PCIN← ALL 1'S 00	PC 01	1	0	PC←00000000; STACKCNT=? INT WAIT=HI INT ENB=? CLOCK IS LO NO INTERRUPTS WAITING

Table 2. Control Point Values

Instructions

NOP (Code 1XXX) No-operation.

When F3 is a 1, control signals C1-C7 are all blocked. Regardless of the other function control bits, the controller performs no other internal instruction than to step the program counter to the next sequential address. This function is provided to control other processors. The single bit F3 determines whether the microinstruction is an internal operation or another processor function. Whenever F3 is a 1, all other bits normally used by the microcontroller (F2 through F0 and the 8-address inputs per slice) are available for other functions.

JUMP (Code 0001) Jump to Instruction given by Address.

This function code provides an unconditional jump to the microinstruction specified by the address input bits A17 through A10.

JTL (Code 0011) Jump on Test Low.

Causes a jump to the instruction specified by the address inputs if the signal on the test input is low. Otherwise the next instruction in sequence is fetched.

PUSH (Code 0100) Push Address Inputs onto Stack.

Causes the word on the address inputs to be pushed onto the control stack and all previous contents of the stack to be pushed down. The next microinstruction in sequence is then fetched. This instruction is normally used to initialize a loop count. If no external memory is connected to the microcontroller, then the stack contains only one element. The "push" operator degenerates into a "store" operator. With a single CD 4036 connected as shown in Figure 4, the stack then contains five elements. The programmer, however, must prevent stack overflow.

TNIJ Test for Non-zero, Increment and Jump.

The contents of the top word of the control stack are tested for zero. If it is non-zero, the value is incremented and returned to the stack, and the address inputs are loaded into the program counter to cause a jump to that address in the microprogram. If the top word of the control stack is zero, the word is popped from the control stack and the next instruction in sequence is fetched. This instruction permits the top word in the control stack to be used as a loop counter, with control passed to the beginning of the loop until the loop counter wraps around to zero, at which time the count is popped from the stack and control passes to the instruction following the TNIJ. When the stack is zero, this operation requires two machine cycles to complete.

JSUB (Code 0110) Jump to Subroutine

The contents of the program counter plus one are pushed onto the control stack, and the word on the address input is loaded into the program counter. This causes a jump to the location specified by the address inputs while saving a return address in the control stack.

Operate Group (Code 0111)

The function or functions performed when the operate group code is encountered depend on the contents of address input bits AI4-AI0. Each bit has a discrete meaning as follows:

AI4 Raise Interrupt Wait Line.

AI3 Set Interrupt Enable Flip Flop.

A12 Reset Interrupt Enable Flip Flop.

$\overline{\text{AI1}}$ Return-Load Top Stack Word into Program Counter.

Note that the JSUB operation stacks the value of the program counter +1 (i.e. the address of the next instruction to be executed).

$\overline{\text{AI0}}$ POP - Pops top word from Stack.

This particular operation is a two-cycle operation. When the POP operation is called for a one cycle delay in the execution of the POP is initiated to give the AI1 control signal, if it is on, a cycle to transfer the top stack element to the program counter. The POP operation may occur without the return operation AI1 being set also, but the POP operation will still require 2 cycles.

Clock System

The Clock system provided on the microcontroller module consists of the Clock generator, the Clock Enable signal, the Clock output, and the Wait line. An external clock is connected to OSC, and it provides the basic timing of the microcontroller.

Clock Generator

The Clock generator uses the basic timing signal to form a pulse train consisting of 3 periods of high level and one of low level. Internal operation of the microcontroller is controlled by the resulting pulse train. While the clock is high, the next microinstruction is fetched from control memory, and all the decoding and settling of the binary values of the various processor control points occur. On the negative clock transition, the relevant microoperations (as shown in Table 2) are performed, and new control values set up. On the positive going clock transition (as shown in Table 2), the program counter is changed, and the next instruction fetch begins. If an instruction causes the stack to change, the change always occurs on the falling edge of the clock. The program counter always changes on the rising edge.

Wait Line

The Wait line input provides a means for a controlled device to delay execution of a microinstruction by pulling this line to ground. The Clock generator will not proceed from the Lo to the Hi output state while the Wait line is grounded. If a controlled device wishes to delay execution of a microinstruction, it must recognize the instruction and ground the Wait line before the leading edge of the clock signal. The Wait line may be released at any time, using the explicit instruction Raise Wait Line.

Clock Enable

The CLK ENB line inhibits CLK OUT signal (the output stays in the high state) when it is grounded, even though internal operations of the microcontroller continue. This provision is primarily for cases in which some instructions are for internal microcontroller use and others are for commanding external functions from controlled devices. In this case, tying the CLK ENB input and the F3 input together causes a CLK OUT pulse to occur, clocking external resisters, etc., only for NOP instructions (F3=1) where bits F2 through F0 and all address inputs are available to control these external devices. When F3 = 0, these lines provide information to the microcontroller itself, and are not available to control external devices. Since F3 = 0, the clock out pulse does not occur and external devices are unaffected.

Interrupt Facility

The microcontroller module is internally capable of handling single level interrupts through the three interrupt signals, INT REQ, INT CLK, and INT WAIT. The INT REQ and INT CLK are a control level and clock signal, respectively, from a requesting device, and an interrupt request on these lines consists of a simultaneous zero on these two inputs. The reason that two lines are required is that the interrupt request line may be the output of another controller's ROM control memory. Each time that controller changes its address counter, noise "glitches" will look like interrupt request signals, and whenever the requesting processor executes an internal control instruction, the interrupt request line may go low for a full cycle. By having the interrupt clock line coming from the clock out of the requesting process, both of these problems are solved. The microcontroller acknowledges an interrupt request immediately by grounding the INT WAIT line. This returned signal represents an unprocessed interrupt and may be used to halt the requesting device.

The interrupt logic contains an interrupt enable flip-flop which may be set or reset under microprogram control. Whenever this flip-flop is set and an unprocessed interrupt is waiting, an interrupt occurs on the next microprogram cycle (unless the previous operation was a two-cycle operation in which case it is delayed one cycle). The interrupt function simulates a JSUB 85 and consists of pushing the program counter contents (the address of the next sequential micro-instruction) onto the stack and forcing the program counter to "85" to force a jump to location 85. The interrupt enable flip-flop is also reset in the process, so that additional interrupts cannot occur until the programmer chooses. The instruction at location 85 is then fetched and executed. As described above, the INT WAIT line remains grounded until it is reset to a high level by the appropriate instruction.

There are no restrictions on when the interrupting process must raise the interrupt request and/or interrupt clock lines. The circuitry used insures that regardless of how slow (or fast) the interrupting process is, exactly one interrupt will result from any coincidence of the low INT REQ and INT CL lines.

This interrupt facility is designed to permit the microcontroller to be seen as an asynchronous device within a larger system, so that it can be requested to perform a service and may delay execution in the larger system until it is ready to accept the task. Depending on the way the interrupt enable flip flop is handled in the microcode, this new request may either interrupt an on-going task or may be held until a previous task is completed. The interrupt facility may also be used within a system to provide transparent handling of some controlled device such as an A/D converter or to provide the basic timing for a repetitive task which is started on each interrupt.

While only a single level interrupt is designed into the micro-controller hardware, extensions to multi-level priority interrupt structures can be developed through microcoding and external priority encoding hardware. Figure 6 shows a convenient way to permit an interrupting process to be able to generate many distinct commands without requiring the interrupted process to use microcode to determine which command to execute, or how, in other words, to implement vectored interrupts. The interrupting process generates the interrupt, and at the same time, places a one or more bit "interrupt code" on predetermined control lines. The interrupted process will immediately ground its wait line, stopping the interrupting process with the above code still on its output control lines. The control lines containing the code are connected via an AND/OR select or 3 state device to the interrupted process' address inputs. When the

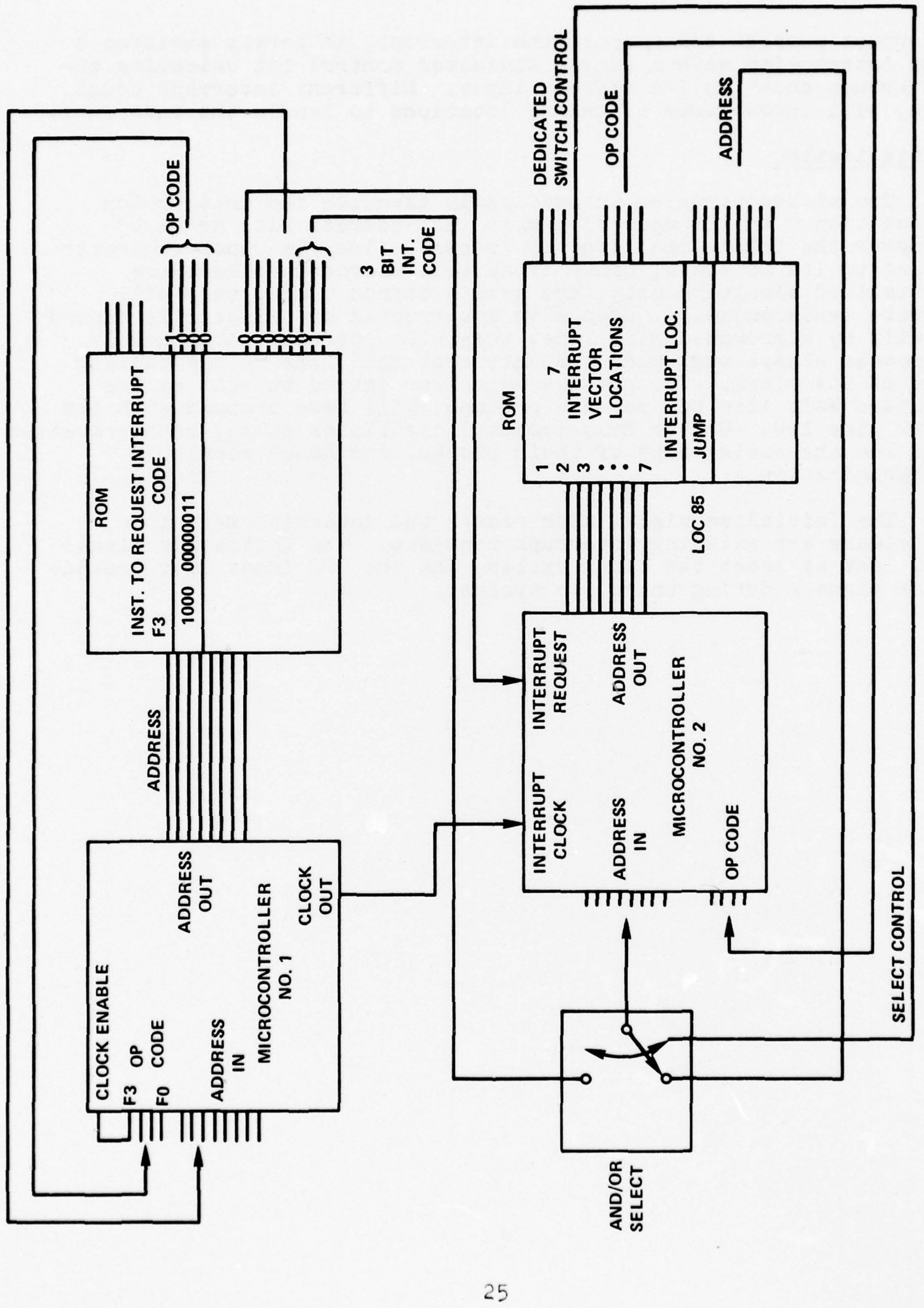


FIG. 6 IMPLEMENTING VECTORED INTERRUPTS

interrupt process can process the interrupt, it merely executes a jump instruction with a single dedicated control bit selecting the "interrupt code" as its address input. Different interrupt codes, then, will cause jumps to unique locations tc handle the interrupt.

Initialization

The microcontroller automatically executes the instruction at location 0 in the control memory (all address bits equal 0) whenever the Initialize input is raised. Also, the clock generator is set to its Lo state, hence if several microcontrollers are initialized simultaneously, and have a common clock, they will operate synchronously. Even when interrupted or temporarily caused to wait by a grounded Wait line, they will operate synchronously. A process always ungrounds its Interrupt Wait line on the falling edge of its clock. The process which was caused to wait by its grounded Wait line but may now proceed, will have stopped with its clock line low. On the next (common) oscillator pulse, both processes will see the rising edge of their clocks, and hence remain in synchronization.

The Initialize signal also resets the Interrupt Wait line, and clears any existing interrupt requests. The Initialize signal must last at least two clock cycles, and the OSC input must provide clock signals during these two cycles.

BASIC TYPES OF MODULES

The purpose of any processing system is to execute some form of an algorithm. If it is desired to configure a system to execute an algorithm out of basic modules, then several types of modules are needed.

Program Memory Module

The algorithm is encoded into a sequence of instructions to be executed by a Control element. The instructions are stored in a program memory module. Since the program must be non-volatile, a Read Only Memory is used.

Read/Write Data Memory

In addition to the program memory, there must be read/write memory for storage and retrieval of data.

Control Element

The Control element reads and executes the program algorithm. The controller performs certain control operations itself, and as required by the program algorithm, delivers commands, control signals, and data to processing modules to perform processing tasks. The microcontroller to be used for this purpose was described in the previous chapters.

Priority Selector

If, in a system, it is necessary to have several modules accessing a single other module, and if these commands can overlap or occur simultaneously in time, then a device to resolve conflicts is needed. The device to accomplish this task is called a priority request arbitrator, priority selector, or port expander.

Figure 7 shows a priority selector used to resolve conflicts when two or more modules can simultaneously access the same process. It operates as follows:

1. Two (or more) modules simultaneously request service from a processing module. The request for service is made by grounding the request line. To prevent noise on the request line from being interpreted as a request, the priority selector does not actually respond to any request until the interrupt clock line also goes low. Figure 7 shows the way in which the microcontroller described in previous chapter is connected to generate the request and clock signals to a processing module or priority selector.

When the Request and Clock lines from one or more modules go low, the priority selector immediately grounds the Wait line to the requesting modules until it can determine which should be given priority.

2. If the receiving PM is busy, then the priority selector must wait. When it is no longer busy then the command and clock signals from the selected requesting PM are passed to the receiving PM. The receiving PM immediately grounds its Wait line, and will leave it grounded until it has finished any processing which must be completed before the requesting PM can proceed.

3. When the receiving PM raises its Wait line, the Wait line to the requesting PM is immediately raised, permitting it to read an answer from the Bus, etc. if the receiving process was an output port, or to remove the data from the bus if the port was an input port.

4. The communication path between the requesting and receiving modules established by the priority selector will not be relinquished until the command line returns Hi, and is Hi when the clock signal goes Lo. At that point, the communication path is disestablished, and another PM given priority. If the command line is still low when the clock goes low, then this is interpreted as another command and is passed to the receiving PM exactly as before. In this way, once communication is established, the requesting process can give as many commands as desired. Note that the requesting process can execute any number of internal commands without relinguishing the communication path so long as it does not lower its clock out signal.

5. The multiplexer can actually be a separate module from the priority selector. The multiplexer then need not have a specific number of bits. With a separate mux, if more bits are needed, another mux module can easily be added. Another approach is to have the priority selector outputs connect to three state control inputs on the requesting and receiving devices, and set up the communication path that way. Using this scheme, no multiplexer's are needed, and there are less interconnecting wires.

Thus far, we have assumed the priority arbitrator is connected between processes which always ground their Wait line when issued a command. If the receiving device is simpler, then a minor refinement is required. Assume as shown in Figure 8 the receiving device is an input port. When the priority arbitrator passes a command and clock signal to the input port, if the port can accept data, it simply opens its latches to receive data, without

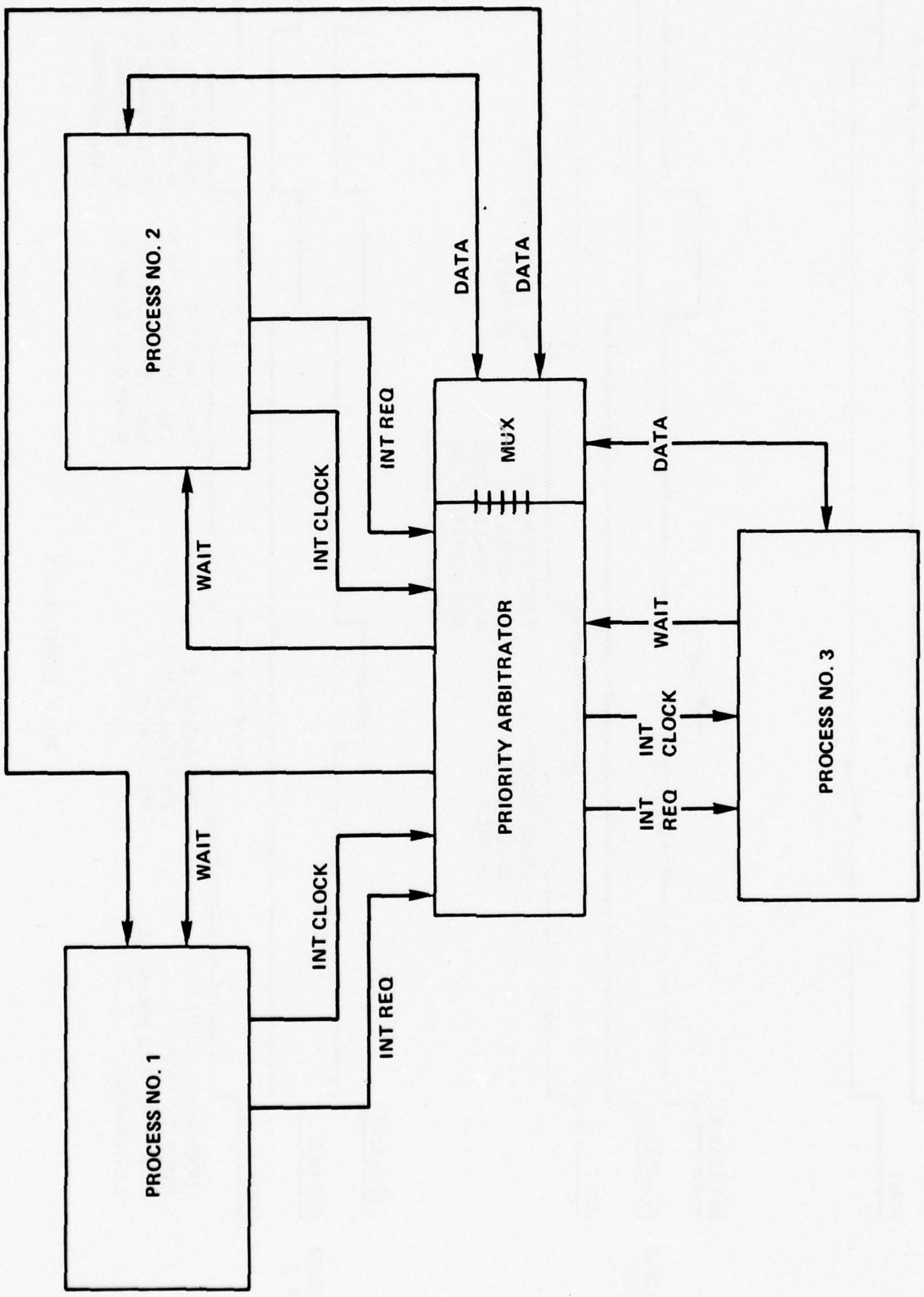


FIG. 7 OPERATION OF THE PRIORITY ARBITRATOR

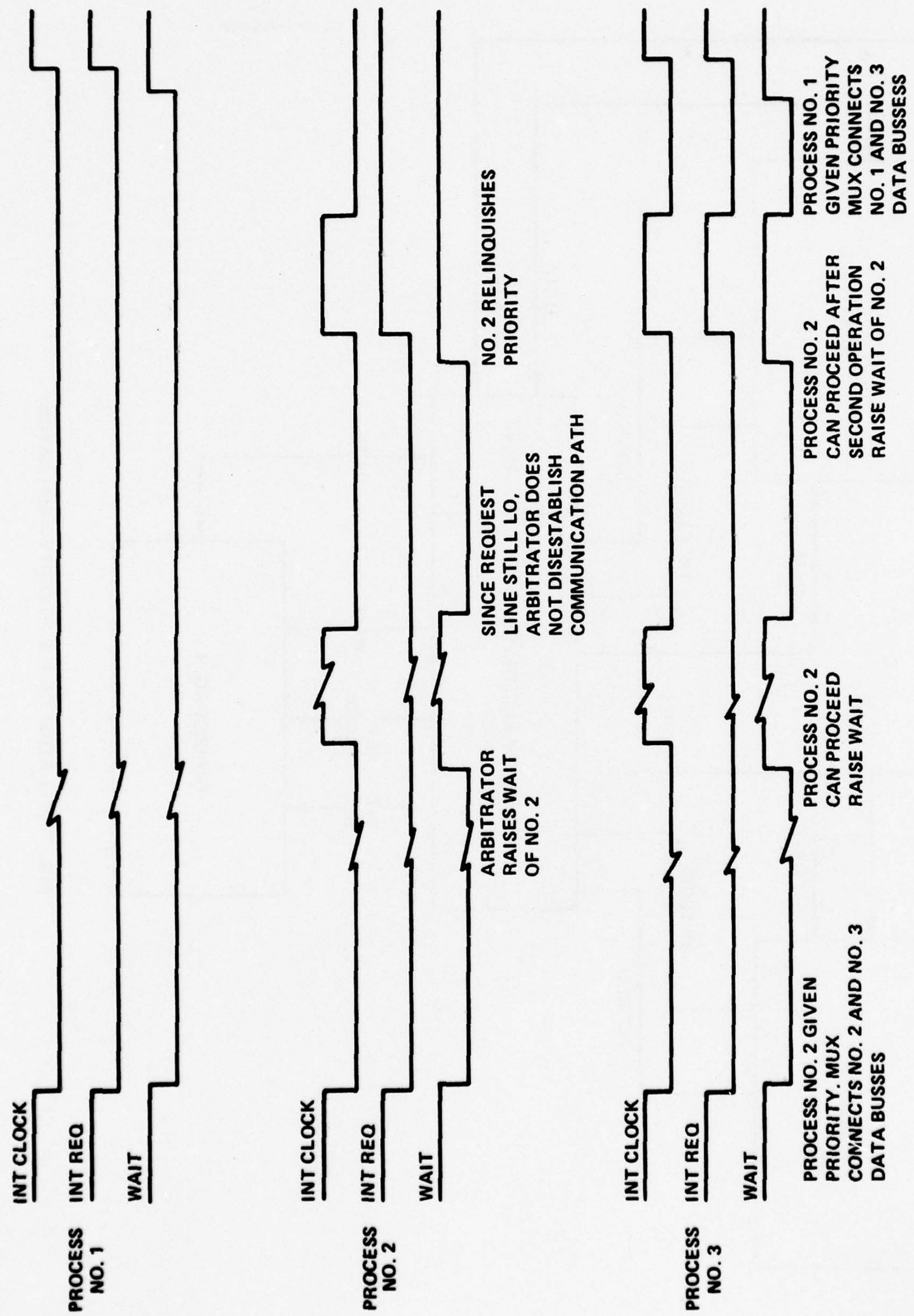

PROCESS NO. 1
INT CLOCK
INT REQ
WAIT
PROCESS NO. 2
INT CLOCK
INT REQ
WAIT
ARBITRATOR RAISES WAIT OF NO. 2
SINCE REQUEST LINE STILL LO, ARBITRATOR DOES NOT DISESTABLISH COMMUNICATION PATH
NO. 2 RELINQUISHES PRIORITY
PROCESS NO. 3
INT CLOCK
INT REQ
WAIT
PROCESS NO. 2 GIVEN PRIORITY. MUX CONNECTS NO. 2 AND NO. 3 DATA BUSSES
PROCESS NO. 2 CAN PROCEED RAISE WAIT
PROCESS NO. 2 CAN PROCEED AFTER SECOND OPERATION RAISE WAIT OF NO. 2
PROCESS NO. 1 GIVEN PRIORITY MUX CONNECTS NO. 1 AND NO. 3 DATA BUSSES

FIG. 7 (CONTINUED)

grounding its Wait line. Without that positive response, the previously described arbitrator would not know when to raise the Wait line of the requesting module. In order to solve this problem, when the priority arbitrator grounds the Command and Clock line to a receiving module, if it does not receive a wait signal from that module within some fixed time, then it will raise the Wait line of the requesting module. This gives the receiving module time to ground its Wait line. This time must be greater than the maximum delay through the receiving module and any other arbitrators connected to the output of the first. This problem could also be solved by always requiring an input port to lower its Wait line for one clock pulse as a positive response, even if it is not busy.

In simple structures where a receiving module is impervious to "glitches", as for example, an output port, then the interrupt clock signals can be tied to ground and not used. Now, however, the priority arbitrator will have no way to know if a second command is being issued unless the Command line goes high. When this occurs, since now the Command line is high while the clock is low, the priority selector will disestablish the communication path, and may reassign priority to another request.

As a specific example of the priority selector, its operation as a memory port expander will be described. Assume, as shown in Figure 9, that two PM's ground their request and Clock lines simultaneously. The priority selector immediately returns a Wait signal to both, and then determines which has the higher priority. When that decision is made, the Command and Clock lines to the selected PM (#2 for example) are grounded, and the mux set to interconnect the buses from the requesting and receiving modules. The receiving module immediately grounds its Wait line. Assume the command code on the command inputs to the receiving module indicate that a read address command is to be executed. The requesting PM will already have the address on the bus, hence the receiving module can either latch the address and begin reading the memory, or address the memory directly with the address on the bus. When the address is no longer needed on the bus, the receiving module raises its Wait line. The priority selector immediately ungrounds the Wait line of the requesting process. The data path previously established is not relinquished and on the next occurrence of the low clock, the priority selector will pass another command to the receiving module. The command code will indicate that the result of the previous read should be placed on the buses. The receiving module will immediately ground its Wait line, and will raise it when the desired memory word is on the bus, The priority selector will unground the requesting processes Wait line, and it will read the result from the bus on the next leading edge of it internal clock.

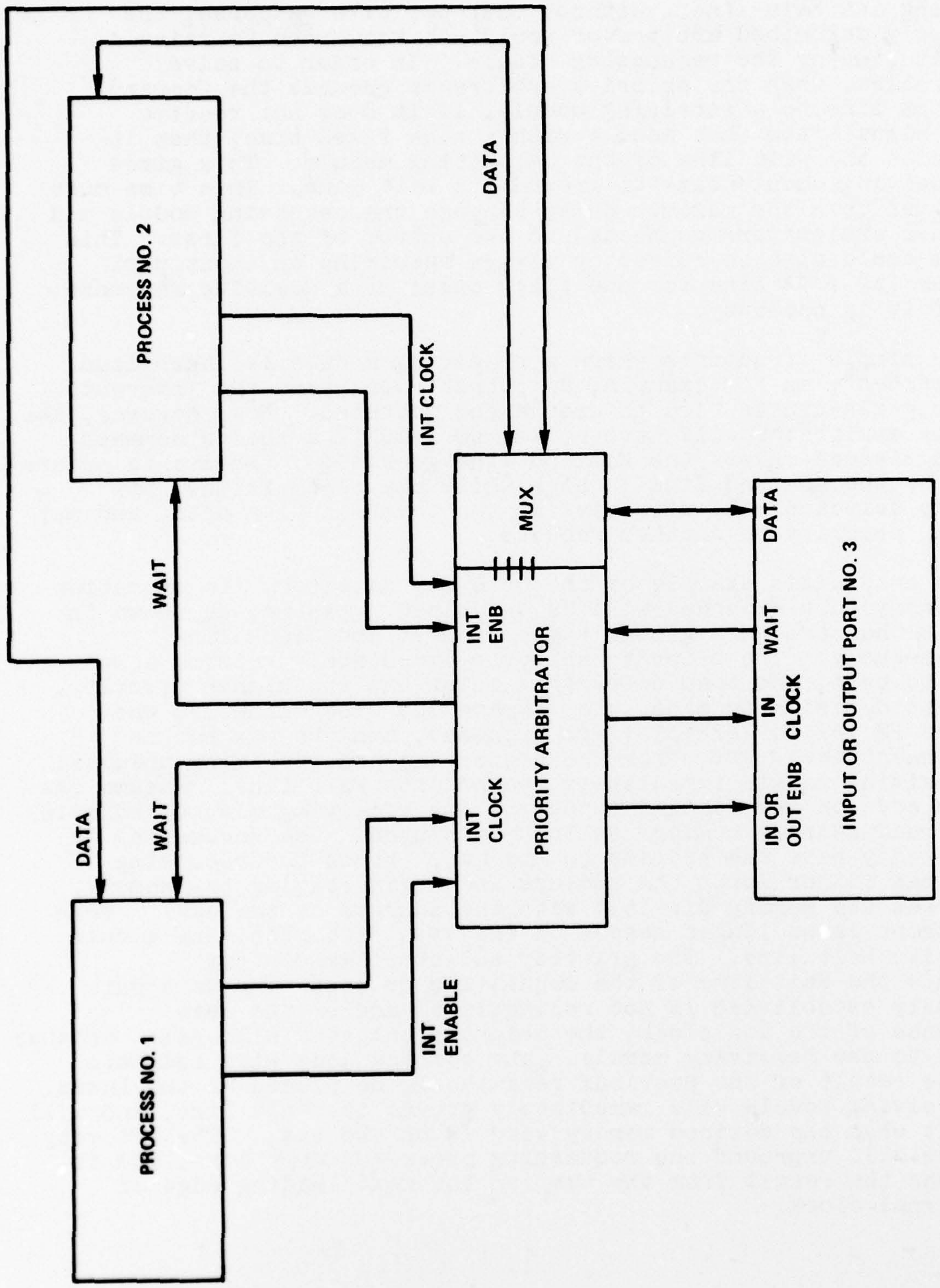


FIG. 8 INPUT OR OUTPUT PORT CONNECTION TO PRIORITY ARBITRATOR

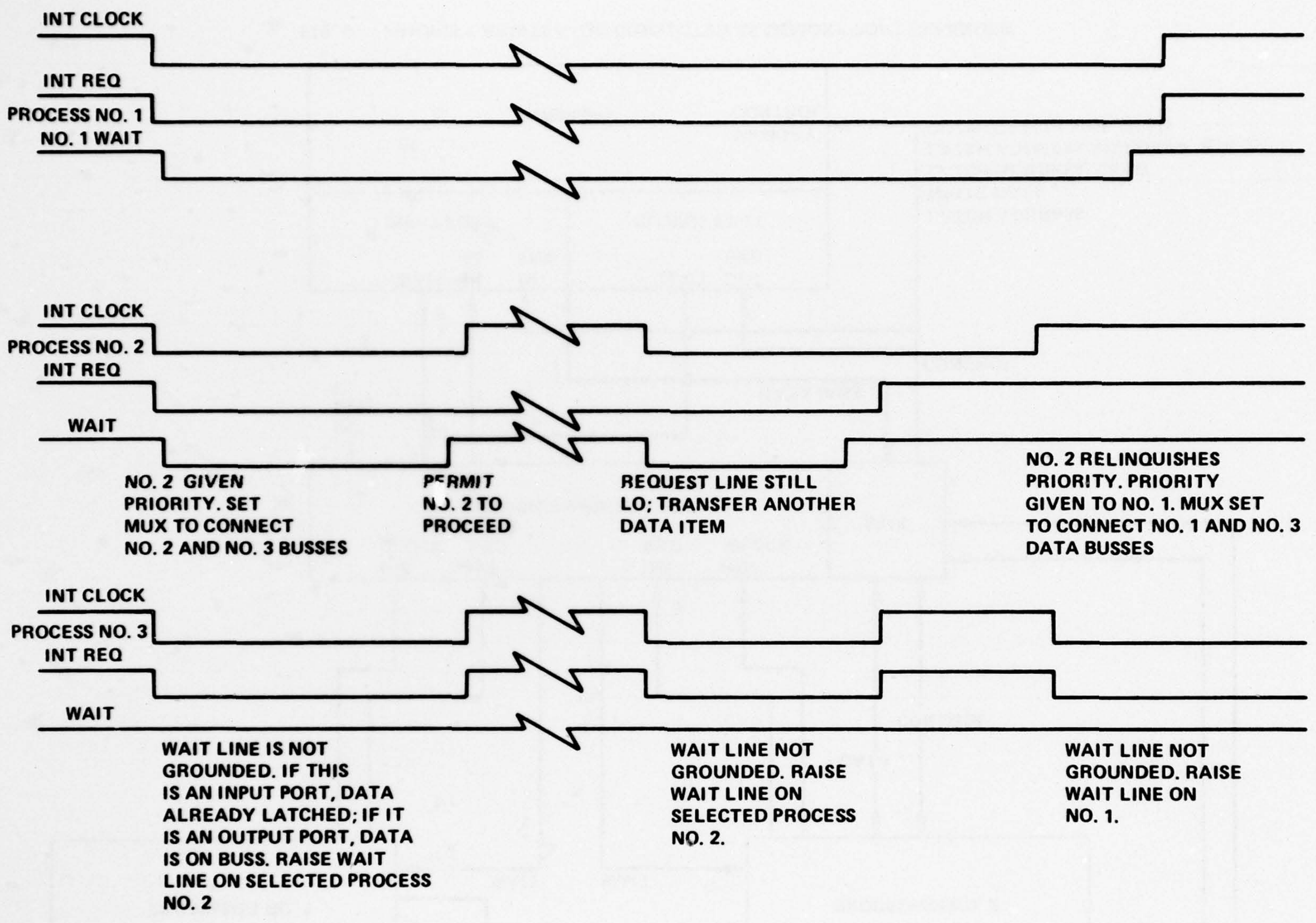

INT CLOCK
INT REQ
PROCESS NO. 1
NO. 1 WAIT
INT CLOCK
PROCESS NO. 2
INT REQ
WAIT
NO. 2 GIVEN PRIORITY. SET MUX TO CONNECT NO. 2 AND NO. 3 BUSSES
PERMIT NO. 2 TO PROCEED
REQUEST LINE STILL LO; TRANSFER ANOTHER DATA ITEM
NO. 2 RELINQUISHES PRIORITY. PRIORITY GIVEN TO NO. 1. MUX SET TO CONNECT NO. 1 AND NO. 3 DATA BUSSES
INT CLOCK
PROCESS NO. 3
INT REQ
WAIT
WAIT LINE IS NOT GROUNDED. IF THIS IS AN INPUT PORT, DATA ALREADY LATCHED; IF IT IS AN OUTPUT PORT, DATA IS ON BUSS. RAISE WAIT LINE ON SELECTED PROCESS NO. 2
WAIT LINE NOT GROUNDED. RAISE WAIT LINE ON SELECTED PROCESS NO. 2.
WAIT LINE NOT GROUNDED. RAISE WAIT LINE ON NO. 1.

FIG. 8 (CONTINUED)

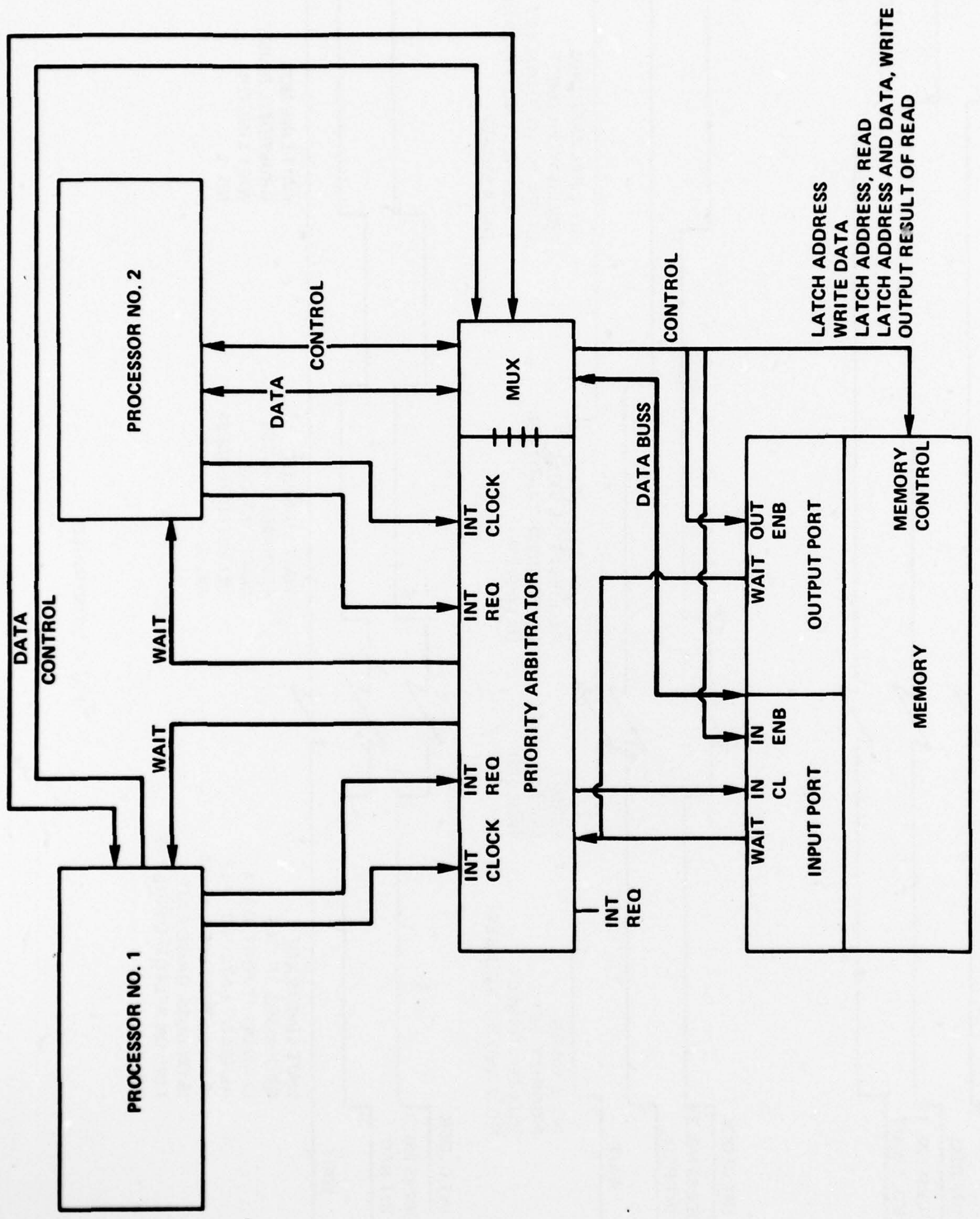


FIG. 9 PRIORITY ARBITRATOR CONNECTED AS MEMORY PORT EXPANDER

In the above example, as long as the requesting process keeps its Command line low (while the clock is low) then any amount of data can be transferred. If the command is a write, then the data and address can be sent simultaneously, and only one command is needed. If the bus is to be multiplexed, then the address can be sent first - followed by the data and write command on the next instruction.

Processing Elements

The "Processing Elements" of a system are the elements which receive commands and data, and map these into some input. Examples of Processing Elements are:

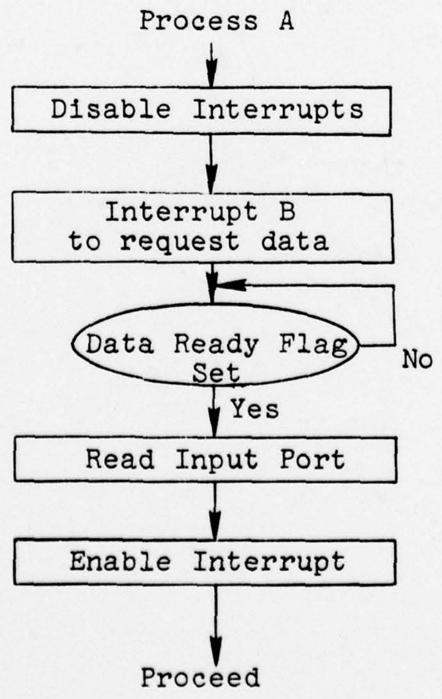
a. A/D or D/A converters

b. Logic, and/or arithmetic elements, etc.

c. Digital filters

d. FFT or correlation processors

A good example of the advantages possible using a microcontroller instead of a single chip microprocessor are seen in the area of communication and synchronization. In a conventional fixed architecture processor, if a processor A requires a predetermined piece of data from an asynchronous processing element B, A would execute the following type of code.



Using the microcontroller, a single instruction

"INTERRUPT B-READ B's BUS"

can easily be implemented by connecting one (or more) of A's arbitrary control lines to B's interrupt inputs, and another to a tri-state buffer which, when Hi, connects B's bus to A's bus. Process B (assuming it is also a microcontroller) immediately (combinatorially) returns a WAIT signal before the rising edge of A's clock, at which time A would latch in the data on the bus. The WAIT signal freezes Processor A, poised ready to read the required data item. B removes the WAIT as soon as the data is on the bus, and A latches the required data item. Numerous instructions been eliminated, no output port is required, and the communication is faster.

Much more powerful forms of interprocess communication can be implemented. It is easy, for example to connect some of A's control lines to implement instructions in Processor A which commandeers (transparent to B) some or all the resources of B. A could read or alter B's memory and input or output ports, or use its arithmetic processor.

EXAMPLE OF A DIRECT EXECUTING COMPUTER IMPLEMENTED WITH THE MICROCONTROLLER

Figure 10 shows a minimum capability digital computer implemented from several elements, including the previously defined microcontroller module. The arithmetic capability is provided by existing RCA Type 4057 Arithmetic/Logic Units. The computer is a "Harvard Type" machine in that memories for its program and its data are logically separated. It operates by direct execution of programmer-generated microcode, rather than by microprogrammed emulation of some virtual machine instruction set, so the Control Memory containing the microcode is also the Program Memory. This approach provides the fastest execution of the program, but the instruction set seen by the programmer consists only of very basic operations. However, in this instance the instruction set approaches that normally found in the smallest minicomputers, so that programming at the microcode level is not an undue hardship. A vertical organization is used in the microcode, so that each instruction is either a sequence control instruction executed by the microcontroller or an execution type instruction performed by other elements. Again this is similar to the instruction set seen in small minicomputers. While this architecture has been investigated primarily to determine the capability of the microcontroller and the 4057 ALU to form a useful computer, it may form a useful data processor in certain small applications. The simple architecture does, however, have several obvious limitations, the most severe of which is its inability to compute or modify a data address. This is a basic requirement in any operations on data arrays.

Because the program and data memories are separate, there is no necessary connection between the data word width and the instruction word width. The data width in the figure is shown as arbitrary, although the four-bit slice width in the 4057 would dispose one to use some multiple of four bits. The instruction word width was arbitrarily chosen as 16 bits. This choice, along with the instruction format chosen, limited the length of the Program Memory to 256 words and the length of the Data Memory to 128 words. These could be expanded by increasing the width of the Program Memory word, but they would generally be adequate for small processing problems.

The instruction set is basically determined by the instruction sets of the microcontroller and of the ALU. The most significant bit of the instruction word is used by the microcontroller to determine whether the operation is a control or execution type. The next three bits are decoded by the microcontroller for control instructions and form Device Select Enable, Memory Cycle Enable, and Read/Write Select control bits for the execute type instructions. The next two bits select one of four test lines for conditional jump instructions, and the remaining ten bits form an address field for all control instructions. For execute instructions the eight least significant instruction bits are used for data memory address

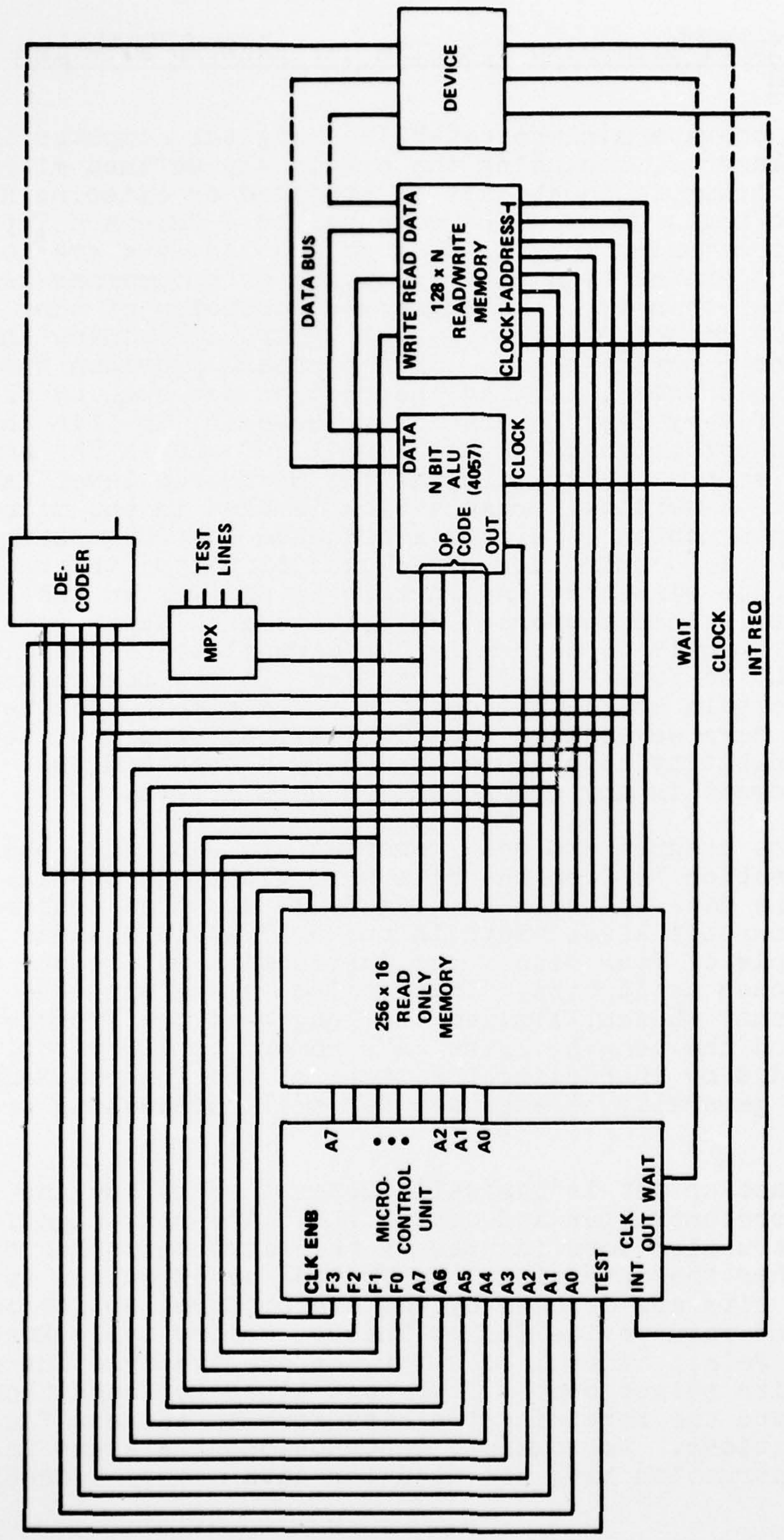


FIG. 10 EXAMPLE OF A DIRECT EXECUTING COMPUTER

information, while the next four higher bits select the ALU operation code. When the Device Select Enable is low, the four least significant bits also select one of sixteen control bits for controlling I/O devices.

The useful instruction set resulting from this architecture is shown in Table 3.

TABLE 3

INSTRUCTION SET OF DIRECT EXECUTING COMPUTER

```
|0 0 0 0 X X X X X X X X X X X X|          NOP
```

No operation is performed. Proceeds to next instruction.

```
0 0 0 1 X X A A A A A A A A A A            JUMP A
```

Unconditional jump to location A

```
|0 0 1 0 N N A A A A A A A A A A|          JTLn A
```

Jump if test line n is Lo, otherwise go to next instruction. n is a number from 0 through 3. The four test lines are connected internally to tests such as Accumulator zero test or sign test, or to discrete tests in I/O device interfaces.

```
|0 1 0 0 X X N N N N N N N N N N|          PUSH N
```

Push the number N onto the control stack and proceed to the next instruction. Normally used to initialize a loop counter.

```
|0 1 0 1 X X A A A A A A A A A A|          TIJ A
```

If loop counter non-zero increment counter and jump to location A. Otherwise pop loop counter from control stack and proceed to next instruction.

```
|0 1 1 0 X X A A A A A A A A A A|          JSUB A
```

Jump to subroutine at location A. Save current address plus one by pushing onto control stack.

```
|0 1 1 1 X X X X X X X S S S S S|          OPGROUP
```

Any set of the following 5 operate group instructions may be combined into one instruction by setting the corresponding bit of the S field.

0 1 1 1 X X X X X X X 0 1 0 1 1	INTE

Set Interrupt Enable flip-flop.

0 1 1 1 X X X X X X X 0 0 1 1 1	INTD

Reset Interrupt Enable flip-flop.

0 1 1 1 X X X X X X X 0 0 0 1 0	POP

Pop word from top on control stack and proceed to the next instruction. Usually used to remove a loop counter from the stack when exit from loop occurs before count reaches zero.

0 1 1 1 X X X X X X X X 0 0 0 0	RETURN

Jump to return location in top word of control stack. Usually used in conjunction with POP for subroutine return.

0 1 1 1 X X X X X X X 1 0 0 1 1	RAISI

Raises the interrupt wait line, permitting an interrupting process to proceed.

1 1 1 X 0 0 0 0 X X X X X X X X	NOP

No operation is performed. Proceeds to next instruction.

1 1 0 1 0 0 0 1 D D D D D D D D	AND D

AND the contents of data word D to the accumulator.

1 1 1 X 0 0 1 0 X X X X X X X X	DEC

Decrement the accumulator by one.

$\boxed{1\ 1\ 1\ X\ 0\ 0\ 1\ 1\ X\ X\ X\ X\ X\ X\ X\ X}$ INC

Increment the contents of the accumulator by one.

$\boxed{1\ 1\ 1\ X\ 0\ 1\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X}$ NEG

Negate the number in the accumulator.

$\boxed{1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ D\ D\ D\ D\ D\ D\ D\ D}$ SUBN D

Subtract the accumulator contents from the contents of data memory word D and place the results in the accumulator.

$\boxed{1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ D\ D\ D\ D\ D\ D\ D\ D}$ ADD D

Add the contents of data memory word D to the accumulator.

$\boxed{1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ D\ D\ D\ D\ D\ D\ D\ D}$ SUB D

Subtract the contents of data memory word D from the accumulator.

$\boxed{1\ 1\ 1\ X\ 1\ 0\ 0\ 0\ X\ X\ X\ X\ X\ X\ X\ X}$ SET

Set the accumulator to all ones.

$\boxed{1\ 1\ 1\ X\ 1\ 0\ 0\ 1\ X\ X\ X\ X\ X\ X\ X\ X}$ CLEAR

Clear the accumulator to all zeros.

$\boxed{1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ D\ D\ D\ D\ D\ D\ D\ D}$ XOR D

Exclusive OR the contents of data memory word D to the accumulator.

1 1 0 1 1 0 1 1 D D D D D D D D	OR D

OR the contents of data memory word D to the accumulator.

1 1 0 1 1 1 0 0 D D D D D D D D	LOAD D

Load the accumulator from data memory word D.

1 1 1 X 1 1 0 1 X X X X X X X X	LEFT

Shift accumulator left one bit.

1 1 1 X 1 1 1 0 X X X X X X X X	RIGHT

Shift accumulator right one bit.

1 1 1 X 1 1 1 1 X X X X X X X X	ROTR

Rotate accumulator contents right one bit.

1 1 0 0 0 0 0 0 D D D D D D D D	STORE D

Store accumulator contents in data memory word D.

1 0 1 1 1 1 0 0 X X X X N N N N	INPUT N

Load accumulator from input device N, N, from 0 through 15.

1 0 1 0 0 0 0 0 X X X X N N N N	OUTPUT N

Place accumulator contents in output device N, N from 0 through 15.

DISTRIBUTION LIST

	Copies
Commander Naval Sea Systems Command Washington, D. C. 20362	
Frank Henry (0333)	1
W. W. Blanc (0333)	1
R. Schuetzler (663C1)	1
Commander Naval Air Systems Command Washington, D. C. 20360	
D. Rosso (AIR 370)	1
A. Stone (AIR 370K)	1
A. Pisano (AIR 370A1)	1
E. Benson (AIR 5330)	1
Commanding Officer Naval Air Development Center Warminster, PA 18974	
D. Russo (2054)	1
J. Howard (205)	1
Defense Documentation Center Cameron Station Alexandria, VA 22314	12
Chief of Naval Operations Washington, D. C. 20350	
J. R. Blouin (NOP 325)	1

DISTRIBUTION LIST (cont.)

	Copies
Project Manager REMBASS Building 443 Fort Monmouth, New Jersey 07703	1
Commander Naval Ocean Systems Center San Diego, California 92132 R. Martinez (Code 4300)	1
Commander Naval Underwater Systems Center Newport, Rhode Island 02840 C. N. Pryor	4
ADTC/AFATL Eglin Air Force Base, Florida 32542 Dr. J. G. Constantine (DLJM)	1